```c
 1
 2  #include <math.h>
 3  #include "tiff.h"
 4  #include "allocate.h"
 5  #include "randlib.h"
 6  #include "typeutil.h"
 7  #include "defs.h"
 8
 9  void error(char *name);
10
11  int main (int argc, char **argv)
12  {
13    FILE *fp;
14    struct TIFF_img input_img, output_img;
15    // define pixel of interest/seed pixel
16    struct pixel s;
17    s.m = 67;
18    s.n = 45;
19    // declare integers i and j for looping later on
20    int i, j;
21    // declare and/or initialize further variables needed
22    int ClassLabel = 1;
23    int NumConnectedPixels;
24    double T = 3;
25
26    if ( argc != 2 ) error( argv[0] );
27
28    /* open image file */
29    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
30      fprintf ( stderr, "cannot open file %s\n", argv[1] );
31      exit ( 1 );
32    }
33
34    /* read image */
35    if ( read_TIFF ( fp, &input_img ) ) {
36      fprintf ( stderr, "error reading file %s\n", argv[1] );
37      exit ( 1 );
38    }
39
40    /* close image file */
41    fclose ( fp );
42
43    /* check the type of image data */
44    if ( input_img.TIFF_type != 'g' ) {
45      fprintf ( stderr, "error:  image must be grayscale\n" );
46      exit ( 1 );
47    }
48
49    // create seg array, initialize all values to zero to start
```

```
50    unsigned int** seg = (unsigned int**)get_img(input_img.width,
         input_img.height, sizeof(unsigned int));
51    for (i = 0; i < input_img.height; i++) {
52        for (j = 0; j < input_img.width; j++) {
53            seg[i][j] = 0;
54        }
55    }
56
57    // call ConnectedSet() function to determine the connected neighbors to
         the seed pixel s
58    ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height,
         ClassLabel, seg, &NumConnectedPixels);
59
60    // display number of connected pixels
61    printf("%d", NumConnectedPixels);
62
63    // for each pixel in input_img.mono, assign either 0 or 255 depending on
         neighbor label within seg
64    for (i = 0; i < input_img.height; i++) {
65        for (j = 0; j < input_img.width; j++) {
66            if (seg[i][j] == ClassLabel) {
67                input_img.mono[i][j] = 0;
68            }
69            else {
70                input_img.mono[i][j] = 255;
71            }
72        }
73    }
74
75    // generate new tiff of proper size, assign to input_img for writing out
         later
76    get_TIFF(&output_img, input_img.height, input_img.width, 'g');
77    output_img = input_img;
78
79    /* open color image file */
80    if ((fp = fopen("output.tif", "wb")) == NULL) {
81        fprintf(stderr, "cannot open file out.tif\n");
82        exit(1);
83    }
84
85    /* write color image */
86    if (write_TIFF(fp, &output_img)) {
87        fprintf(stderr, "error writing TIFF file %s\n", argv[2]);
88        exit(1);
89    }
90
91    /* close color image file */
92    fclose(fp);
93
```

```
 94      /* de-allocate space which was used for the images */
 95      free_TIFF(&(input_img));
 96      free_img((void*)seg);
 97
 98      return(0);
 99  }
100
101  void error(char *name)
102  {
103      printf("usage:  %s  image.tiff \n\n",name);
104      printf("this program reads in a 24-bit color TIFF image.\n");
105      printf("It then horizontally filters the green component, adds noise,  ⏎
              \n");
106      printf("and writes out the result as an 8-bit image\n");
107      printf("with the name 'green.tiff'.\n");
108      printf("It also generates an 8-bit color image,\n");
109      printf("that swaps red and green components from the input image");
110      exit(1);
111  }
112
113
```

```c
1  #pragma once
2  #include <stdlib.h>
3
4  struct pixel {
5      int m, n;          // m = row, n = col
6  };
7
8  void ConnectedNeighbors(struct pixel s, double T, unsigned char** img, int  ⏎
     width, int height, int* M, struct pixel c[4]);
9
10 void ConnectedSet(struct pixel s, double T, unsigned char** img, int width, ⏎
      int height, int ClassLabel, unsigned int** seg, int* NumConPixels);
```

```c
1  #include "defs.h"
2
3  void ConnectedNeighbors(struct pixel s, double T, unsigned char** img, int  ⮐
      width, int height, int* M, struct pixel c[4]) {
4      *M = 0;
5      if (s.m − 1 >= 0 && abs(img[s.m][s.n] − img[s.m − 1][s.n]) <= T) {
6          c[*M].m = s.m − 1;
7          c[*M].n = s.n;
8          (*M)++;
9      }
10     if (s.m + 1 < height && abs(img[s.m][s.n] − img[s.m + 1][s.n]) <= T) {
11         c[*M].m = s.m + 1;
12         c[*M].n = s.n;
13         (*M)++;
14     }
15     if (s.n − 1 >= 0 && abs(img[s.m][s.n] − img[s.m][s.n − 1]) <= T) {
16         c[*M].m = s.m;
17         c[*M].n = s.n − 1;
18         (*M)++;
19     } if (s.n + 1 < width && abs(img[s.m][s.n] − img[s.m][s.n + 1]) <= T) {
20         c[*M].m = s.m;
21         c[*M].n = s.n + 1;
22         (*M)++;
23     }
24 }
```

```c
1  #include "defs.h"
2  #ifdef _MSC_VER
3  #include <malloc.h>
4  #endif
5  #include <assert.h>
6
7  void ConnectedSet(struct pixel s, double T, unsigned char** img, int width, ⏎
       int height, int ClassLabel, unsigned int** seg, int* NumConPixels) {
8      *NumConPixels = 0;
9      struct pixel* B = alloca(sizeof(struct pixel) * width * height);
10     unsigned int lenB = 0;
11     unsigned int M = 0;
12     struct pixel c[4];
13
14     if (B == NULL)
15         return;
16
17     B[lenB++] = s;
18
19     while (lenB > 0) {
20
21         assert(lenB < width* height);
22         s = B[lenB - 1];
23         --lenB;
24
25         ConnectedNeighbors(s, T, img, width, height, &M, c);
26
27         if (seg[s.m][s.n] == 0) {
28             (*NumConPixels)++;
29         }
30
31         seg[s.m][s.n] = ClassLabel;
32
33         for (int i = 0; i < M; i++) {
34             if (seg[c[i].m][c[i].n] == 0) {
35                 B[lenB++] = c[i];
36             }
37         }
38     }
39 }
```